JUKES CANTOR

In phylogeny, the observed substitutions in a multiple sequence alignment are underestimating the true number of substitutions.
In this project, you will show how the simplest model of substitutions can try to estimate the likelihood of substitutions.

Report

Carefully read the whole text and — if required — read the lectures about the course *Omics and BioInformatics* and the corresponding Wikipedia pages (about models of DNA evolution).

Manuscript

Write your report like a scientific article with the classic sections:

- Introduction
- Material & Methods
- Results
- Discussion & Conclusion
- References

The report must be written in English.
The report must contained a title with the author's name.
All the figures must have a legend.
All the figures and references must be called in the text.
Take care to add correct references (scientific articles and NOT websites).

The report — saved as a PDF file — will be uploaded in the corresponding MOODLE page.

Supplementary Data.

The supplementary data must contained all the Python functions required to

run your project. This is the only place where the code must be inserted. It will be uploaded as a separate Python file in the corresponding MOODLE page.
No test must be added, ONLY the functions without any function calls.

Programming

We have split the whole script in various functions to simplify the implementation. Thus, it is very advised to re-use the code/functions to have the most compact possible final script.
The only functions authorized for this project are the built-in functions of Python 3.x (see the official documentation).

Two specific functions are allowed in your script.

Function `random.randint(..)`

The random generator used throughout this page is the function `randint(..)` of the module `random`. To use it, you have to import it

```
import random
```

and then, to call the function `randint` of the module `random`, you have to type...

```
value = random.randint(0,5)
```

This code generates a random value in the range of $[0, 5]$

Module `math`

The other mathematical functions required are in the math module...

```
import math
print(math.log(3))
```

> **Note**: All the other functions from other modules are not authorized in the code.

Part I - Generating a random sequence

Exercise 1.1.

Write a function `alphabet(name)` taking one argument `name`. This function returns the one-letter alphabet used in bioinformatics as a *String* according to its **name**.

name	alphabet
nucleic	The classic alphabet of one-letter code of the 4 nucleotides
protein	the classic alphabet of one-letter code of the 20 amino-acids
iupac_nucleic	The IUPAC alphabet for nucleotides
iupac_protein	The IUPAC alphabet for amino-acids

> **Note**: To get the correct and *official* alphabets, go to the website of the IUPAC (International Union of Pure and Applied Chemistry).

Example of use:

```
# Write your code here
def  alphabet(type):
  # TODO

# Test
nuc = alphabet('nucleic')
print(nuc) # Expected result: 'acgt'.
inuc = alphabet('iupac_nucleic')
print(inuc) # Returns a String containing all the IUPAC nucleotide c
```

Exercise 1.2. Function randseq(...)

Write a function `randseq(num,alpha)` taking two arguments corresponding to the length of the sequence and the alphabet. This function randomly generates

a sequence of length **`num`** from the alphabet **`alpha`**.

```
import random

def  randseq(num,alpha):
  # TODO
  return seq

# Tests
anuc = alphabet('nucleic')
test = randseq(20,anuc)
print(test,len(test)) # Something like 'tgaccaggtttctagcgtct' 20
print(randseq(34,alphabet('protein')))
```

Part II - Generating the substitutions

Exercise 1.3. Function hamming(...)

Write a function `hamming(seq1,seq2)` computing and returning the Hamming distance between two sequences of type *String*.

```
def  hamming(seq1,seq2):
  #TODO

# Test
d = hamming('acgtacgt','aggtacga')
print(d) # Expected result: 2
```

Exercise 1.4. Function mutate(...)

Write a function `mutate(seq,num_subs)` computing a sequence mutated `num_subs` times from the input sequence `seq`. The mutated sequence must be returned as a *List* (and not as a *String*).

- Initialize the mutated sequence by converting the `seq` into a *List* using the builtin Python function `list(..)`.
- For each substitution

    - Randomly choose the location where the mutation will occur using random.randint(...) function.
    - Get a new single nucleotide — the substitution — by using randseq(...)
    - Set the new nucleotide into the mutated sequence
    - Repeat the operations `num_subs` times.
- Return the Hamming distance between `seq` and the mutated sequence.

> **Example**: Step-by-step execution of `mutate(..)` with the sequence 'acgtacg' and a number of substitutions of `5`.

```
h = mutate('acgtacg',5)
print(h)
```

Here is an example of a step-by-step execution of the script above.

cycle	seq	mutated	Event
Init	acgtacg	['a', 'c', 'g', 't', 'a', 'c', 'g']	—
1	acgtacg	['a', 'c', 'a', 't', 'a', 'c', 'g']	g_2 -> a
2	acgtacg	['a', 'c', 'c', 't', 'a', 'c', 'g']	a_2 -> c
3	acgtacg	['a', 'c', 'c', 't', 'a', 'c', 't']	g_6 -> t
4	acgtacg	['a', 'c', 'c', 'c', 'a', 'c', 't']	t_3 -> c
5	acgtacg	['a', 'c', 'c', 'c', 'a', 't', 't']	c_5 -> t

The function returns — in this specific run — a Hamming distance of 4.

> **Note**: The results of the step-by-step execution may vary because of the randomness of the substition process.

Run the function mutate(...) for a random sequence of 1000 nucleotides and a substitution number of 1000.

Use the function `experiments(le,su,nb)` for repeating the experiment `nb` times and returning all the Hamming distances in a *List*.

Example of use:

```
def experiments(le,su,nb):
  # Repeat the experiments `R` times
  v = []
  for i in  range(nb):
    seq = randseq(le)
    v.append( mutate(seq,su) )
  return v

# Main
L = 1000 # Seq Length
M = 1000 # Number of substitutions
R = 10 # Number of experiments
results = experiments(L,M,R)
print(results)
```

Part III - Statistics and Jukes Cantor Model

Exercise 1.5: Basic Statistics - Mean

Write a function `mean(data)` computing the mean of the *List* `data`.

```
m = mean([1,2,6])
print(m) # Expected result 3
```

Exercise 1.6: Basic Statistics - Variance

Write a function `variance(data)` computing the variance according to **Welford's method**.

Exercise 1.7: Basic Statistics - Standard Deviation

Write a function `std(data)` returning the standard deviation of the *List* `data`.

> **Note**: Use the module `math` for the square root.

Exercise 1.8: Distance according to Jukes-Cantor Model

The genetic distance according to the Jukes Cantor model (1969) can be deduced from the model and the formula is:

$d = \frac{-3}{4} * ln(1.0 - p * \frac{4}{3})$ (source Wikipedia)

Write a function `distanceJC69(means, L)` where `means` is the mean calculated from a series of experiments and `L` is the sequence length. This function returns a number corresponding to the genetic (corrected) distance according to Jukes Cantor model (1969).

Example of use:

```
# Example of Results obtained from
# the function `mutate(..)` repeated 5 times.
v = [101,107,99,105, 109]
m = mean(v)
L = 700 # Sequence length
d = distanceJC69(m,L)
print(d) # genetic distance of a mutated sequence of length 700.
```

Part IV - Plots

For this project, we want to plot two curves allowing the comparison of the true number of substitutions, the number of observed substitutions (Hamming) and the number of substitutions calculated by the Jukes Cantor model from the Hamming Distances.

Generating Distance Data from a random sequence

For that, we need to compute for a random nucleic sequence of length 1000. The Hamming and JC69 distances for all the substitutions in the range of [100,2000] with a step of 200.

substitutions number	Hamming	distance JC69

substitutions number	Hamming	distance JC69
100	must be computed	must be computed
120	must be computed	must be computed
140	must be computed	must be computed
...	...	...
1600	must be computed	must be computed
1800	must be computed	must be computed
2000	must be computed	must be computed

Write a function `generate(le,nb,xx)` where `le` is the sequence length, `nb` is the number of experiments, and `xx` is a *List* containing a variable number of substitutions. This function returns a *List* containing two elements:

- A *List* of Hamming Distances calculated from the *List* `xx` .
- A _List of JC69 Distances calculated from the *List* `xx` .

```
# Init
L = 1000
R = 10
x = list(range(100,2200,200))
data = generate(L,R,x)
print(data[0]) # Contains a List of the Hamming Distances
print(data[1]) # Contains a List the JC69 Distances
```

Plotting

In Python, we use the module `matplotlib` for all kind of sketches. Here is a simple example of a curve.

```
import matplotlib.pyplot as plt
import math

# Generate `n` values from 0 to 2PI
```

```
n = 40
step = 2 * math.pi / n
xx = [0] * n
yy = [0] * n
for i in  range(n):
  xx[i] = i * step
  yy[i] = math.sin(xx[i])

# Draw plots
# `f(x) = sin(x)`
plt.plot(xx, yy, label='sin(x)')

# Configuration
plt.xlabel('x')
plt.ylabel('f(x)')

plt.title("Example of Plot")
plt.legend()
plt.show()
```

The first step is to import the module `pyplot` of the package `matplotlib` and we rename it as `plt` (just for convenience because pyplot is a long word).

```
import matplotlib.pyplot as plt
```

The main function is `plt.plot(..)` accepting the X- and Y-coordinates of your curve. You can add a label to this curve using the keyword argument `label='my title'`.
Finally, do not forget the function `plt.show()` to display your curve.

> **Note**: The other functions are for the legend and the labeling of axes.

> **Note**: If you want to display more curves in the same figure, just add other `plt.plot(..)` lines in your code.

Plots

In this project, we need to calculate X-Y plots of:

- True substitutions number *vs* Hamming Distances

- True substitutions number *vs* JC69 Distances

By modifying the previous script using `matplotlib` , plot them in the same figure.

Describe and discuss the two curves.

- What are the shapes of the 2 curves? Discuss each of the two curves?
- What happens when the number of substitutions is small? high?
- Is the JC69 model good for our random sequence?
- etc.

Feel free to try with other parameters (use more points, various numbers of substitutions and/or by varying other points) if you think it will improve your discussion/manuscript.